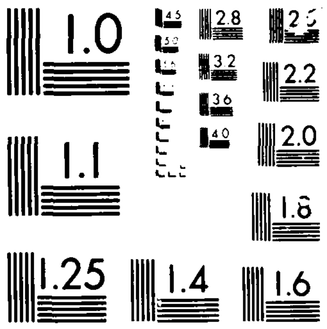
AD-A168 627   MESH-CONNECTED PROCESSOR ARRAYS FOR THE TRANSITIVE   1/1
CLOSURE PROBLEM(U) STANFORD UNIV CA INFORMATION SYSTEMS
LAB   S K RAO ET AL. DEC 85 AFOSR-TR-86-0264
UNCLASSIFIED   AFOSR-83-0228   F/G 9/3   NL



MICROCOP' CHART

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

AD-A168 627

1a REPORT SECURITY CLASSIFICATION: Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY: NA	3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for Public Release; Distribution Unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE: NA	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 86-0264
6a NAME OF PERFORMING ORGANIZATION: Stanford University; 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION: AFOSR/NM
6c ADDRESS (City, State and ZIP Code): Department of Electrical Engineering, Durand 117, Stanford, CA 94305	7b ADDRESS (City, State and ZIP Code): Bldg. 410, Bolling AFB, DC 20332-6448
8a NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR; 8b OFFICE SYMBOL (If applicable): NM	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: AFOSR 83-0228
8c ADDRESS (City, State and ZIP Code): Bldg. 410, Bolling AFB, DC 20332-6448	10 SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: 6.1102F; PROJECT NO.: 2304; TASK NO.: A5; WORK UNIT NO.:
11 TITLE (Include Security Classification): Mesh-Connected Processor Arrays for the Transitive Closure Problem.	
12 PERSONAL AUTHOR(S): S.K. Rao, Todd Citron and Thomas Kailath	
13a TYPE OF REPORT: reprint; 13b TIME COVERED: FROM 84 TO 85	14 DATE OF REPORT (Yr., Mo., Day): Dec. 1985; 15 PAGE COUNT: 6
16 SUPPLEMENTARY NOTATION: IEEE 24th Conference on Decision and Control, Ft. Lauderdale, Florida, December 1985.	
17 COSATI CODES: FIELD / GROUP / SUB GR: XXXXXXXXXXXX	18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number): mesh-connected processor arrays, transitive closure problem, systolic architectures, matrix multplication, array, iterative algorithm.

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The main purpose in this paper is to lay a theoretical foundation for the design of mesh-connected processor arrays for the transitive closure problem. Using a simple path-algebraic formulation of the problem and observing its similarity to certain well-known smoothing problems that occur in digital signal processing, we show how to draw upon existing techniques from the signal processing literature to derive regular iterative algorithms for determining the transitive closure of the graph. The regular iterative algorithms that are derived using these considerations, are then analyzed and synthesized on mesh-connected processor arrays. Among the vast number of mesh-connected processor arrays that can be designed using this unified approach, the systolic arrays reported in the literature for this problem are shown to be special cases.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED [✓] SAME AS RPT ☐ DTIC USERS ☐	21 ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: Brian W. Woodruff	22b TELEPHONE NUMBER (Include Area Code): (202) 767-4939; 22c OFFICE SYMBOL: AFOSR/NM

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

EE 24th Conf. on Decision and Control

D & C Conf 12/85 Ft. Lauderdale, Florida

A-1

MESH-CONNECTED PROCESSOR ARRAYS FOR THE TRANSITIVE CLOSURE PROBLEM*

AFOSR-TR- 86-0264

Sailesh K. Rao, Todd Citron and Thomas Kailath
Information Systems Laboratory
Stanford University
Stanford, CA 94305.

Abstract

The main purpose in this paper is to lay a theoretical foundation for the design of mesh-connected processor arrays for the transitive closure problem. Using a simple path-algebraic formulation of the problem and observing its similarity to certain well-known smoothing problems that occur in digital signal processing, we show how to draw upon existing techniques from the signal processing literature to derive regular iterative algorithms for determining the transitive closure of the graph. The regular iterative algorithms that are derived using these considerations, are then analyzed and synthesized on mesh-connected processor arrays. Among the vast number of mesh-connected processor arrays that can be designed using this unified approach, the systolic arrays reported in the literature for this problem are shown to be special cases.

I.Introduction

An interesting problem that often arises in various graph theoretic applications is the transitive closure problem. Here the objective is to determine whether or not there exists a path between any pair of nodes in a directed graph. This problem has been addressed in the context of neural modeling, routing, decision making, circuit simulation, transportation problems *etc.* and a solution to this problem can be easily adapted to determine the shortest(longest) path between two nodes in a weighted directed graph, the number of paths between these two nodes (if the graph is acyclic), for enumerating the paths themselves, for determining the set of strongly connected components in the graph, for determining the minimum spanning trees of the graph, for determining the bridges in a graph and so on [5].

In the usual formulation of the problem, the graph is given in terms of its adjacency matrix **A**, where $a_{ij} = 1$ if there is a directed arc in the graph from node i to node j, and 0 otherwise. Then, the transitive closure of the graph is requested in the form of the matrix **T** where $t_{ij} = 1$ if there is a directed path in the graph from node i to node j, and 0 otherwise.

Systolic Architectures for solving this problem on a mesh connected array of processors have been derived in Guibas, Kung and Thompson [1] and recently in Kung and Lo[2]. However, using the recently developed methodology in [3,4], we can show that there are a vast number of systolic configurations for solving the problem. Indeed, any systolic array solution for the matrix multiplication problem can be simply adapted to solve the transitive closure problem; it has been shown in [4] that there are 27 different array configurations with strictly *nearest neighbour* interconnections for matrix multiplication. Thus, there are that many array configurations for solving the transitive closure problem as well. The methodology described in [3,4,17] for deducing these array configurations consists of the following steps:

i. Determine a well-structured iterative algorithm (what is known as a *regular iterative algorithm* in [3,4,17]) for solving the problem,

ii. Analyze the algorithm and obtain various implementation-independent properties of the algorithm, including a valid computational schedule, and

iii. Map the algorithm onto a mesh-connected processor array for which the computed schedule can be applied.

In this paper, we shall mainly concentrate upon the first of these steps. The other two steps are just applications of the techniques developed in [3,4,17] and we shall only briefly discuss them.

Organization of the Paper: The first step above is addressed in Section II. Here we shall show how to derive regular iterative algorithms for solving the transitive closure problem. In particular, we shall obtain a mathematical formulation of the problem so as to expose its relation to a certain well known problem in signal processing. Thereupon, we show how to use some established techniques in that field for obtaining regular iterative algorithms for solving the problem. The resulting algorithms comprise those given in [1] and [2] as special cases. Once a Regular Iterative Algorithm is found, one can apply the techniques outlined in [3,4,17] to determine implementations for it. Some examples are given at the end of this section, though the actual details can be found in [19].

In the interest of brevity, all the theorems and procedures given in this paper are stated without proof. The interested reader is referred to a longer version of this paper[19], for the nitty-gritty details.

II. Regular Iterative Algorithms for the Transitive Closure Problem

The design of regular iterative algorithms for the transitive closure problem is discussed in this section. An iterative algorithm is said to be regular if it can be expressed as a set of functional relations as follows: For $j = 1$ to V do:

$$x_j(I) = f_{I,j}(x_1(I-D_{1,j}), x_2(I-D_{2,j}), \cdots, x_V(I-D_{V,j}), \cdots)$$

where V is some constant integer that determines the number of indexed variables used in the algorithm and I is an S-dimensional integer vector that ranges over a prespecified set of points known as the index points. The index points span a certain S-dimensional region known as the *Index Space*. In addition, for a regular iterative algorithm, the vectors $D_{i,j}$ are required to be *independent* of the index point I and the 'extent' of the Index Space. More on regular iterative algorithms and their properties can be found in [3,17,18].

In the usual path-algebraic formulation of the transitive closure problem, one determines the transitive closure of the graph by solving a set of linear equations[6-8] over a semi-ring. Using this formulation, many known algorithms for solving linear equations, such as Gaussian elimination, Gauss-Siedel iteration or Gauss-Jordan elimination can be simply modified to solve the transitive closure problem. The resulting algorithms for the transitive closure problem are better known in the graph-theoretic literature as Carre's algorithm[7], Ford-Fulkerson's algorithm[9] and Warshall's algorithm[10] (or Floyd's algorithm [11]) respectively. Furthermore, these algorithms can be written in a regular

*This work was supported in part by Lockheed Missiles and Space Company, under Contract FBSAAS930F, by NASA Headquarters, Center for Aeronautics and Space Information Sciences (CASIS) under grant NAGW-419, and by the US Air Force Office of Scientific Research, Air Force Systems Command, under Contract AFOSR83-0228

iterative format, and thus, using the techniques given in [3,4], one can easily generate a multitude of mesh connected processor implementations for the transitive closure problem.

Even though there is such a striking relationship between the algorithms given in [6-10] and certain well-known procedures for solving linear equations, there does not seem to be an obvious correspondence between the transitive closure algorithm of Guibas, Kung and Thompson[1] and any known linear algebraic procedure. In this section, we shall show how one can obtain an alternative formulation of the transitive closure problem that bears a marked resemblance with certain well known problems in digital signal processing. Thereupon, using certain known signal processing techniques, we shall derive several regular iterative algorithms for solving the transitive closure problem. These algorithms have no direct counterparts in linear algebra, but have close relationships with known algorithms in recursive filtering. Furthermore, Guibas et. al.'s algorithm is a special case of the algorithms that are derived in this section using this approach. Curiosly enough, even Warshall's algorithm can be derived as a special case of these algorithms.

Not only can the transitive closure problem be formulated as a set of linear equations, it can also be determined by solving a set of Lyapunov equations, or by solving a set of matrix quadratic equations. For the purpose of this paper, the formulation of the problem as a set of quadratic equations is of interest, since this formulation exposes the similarity of the transitive closure problem with certain image smoothing problems. In this formulation, the element in the i^{th} row and the j^{th} column of the transitive closure matrix of the graph, T, is expressed as

$$\mathbf{t}_{ij} = \mathbf{a}_{ij} \vee \left\{ \bigvee_{\substack{k=1 \\ k \neq i,j}}^{N} \mathbf{t}_{ik} \wedge \mathbf{t}_{kj} \right\} \qquad (1)$$

where '$\vee$' represents the logical *OR* operation and '$\wedge$' represents the logical *AND* operation. The above expression is merely an algebraic statement of the obvious fact that a path from node i to node j exists if and only if there exists either an edge from i to j (i.e. $\mathbf{a}_{ij} = 1$) or if there exists a path from i to k and k to j for some k.

The recursive computation of a matrix as suggested by eqn.(1) is similar to certain computations that arise in the recursive smoothing of digital signals [12]. Thus, it is only natural to ask whether the techniques used in signal processing for solving such problems can be adapted for the transitive closure problem. The main objective in this section is to show that this is indeed so.

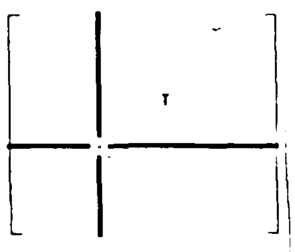


Fig.1. The cyclic nature of the definition in eqn. (1). The darkened lines indicate the entries of T that have to be computed before the typical element marked $\times$ can be computed. In the image processing literature [13-14], this would be referred to as the *output mask* of the recursive filter.

A close examination of the system of equations defined by eqn. (1) reveals that it is impossible to solve the transitive closure problem by computing the entries of the **T** matrix using eqn. (1), no matter how we order the computation. This is because of the cyclic definition of the elements of the matrix in that, *e.g.*, t_{ij} is dependent upon say, t_{ik}, but t_{ik} is dependent upon t_{ij} for any $k \neq j$ (See Fig.1). This 'nonrecursible' situation often arises typically in the recursive smoothing of (noisy) digital signals wherein it is usually handled using one of two main approaches:

i. *Extended Filtering:* For the transitive closure problem, this technique would correspond to solving a simpler 'recursible' problem for an extended graph.
ii. *Forwards-Backwards Filtering:* In this approach, one would attempt to recast eqn.(1) in a recursible form such that the problem can be solved by (possibly repeated) 'forward' and 'backward' passes through the adjacency matrix.

We shall mainly elaborate upon the first approach in the rest of this section. Thus, we shall show that by solving a recursible problem for a suitably extended graph, we can obtain the transitive closure of the given graph. In addition, we shall also characterize a family of extended graphs for which this property holds. For a graph with E directed arcs, this family has more than 2^{E} members. For this family of extensions, we shall present an algorithm that can be applied on any member, to obtain the transitive closure of the given graph. We shall show that this algorithm, when applied to one particular member of this family, can be related to the algorithm due to Guibas, Kung and Thompson [1]. We shall also show that the algorithm when specialized to yet another member of this family, after suitable 'folding' of the computation, is exactly the algorithm due to Warshall[10] which was recently implemented on a systolic array in [2]. It is interesting to note that algorithm in [10], though it can be derived as a special case of the extended graph approach, really solves the problem by performing recursive computations using forward and backward passes through the adjacency matrix.

Before, we proceed to postulate the extended graph problem *per se*, we need to define the concept of a *t–path* (see e.g Ullman[5]).

Definition: t-path: A path from node i to node j (or node j to node i) is said to be a t-path if every node n, that lies along this path, excluding the end points, has index less than t, i.e., it satisfies $n < t$.

For the present, we shall be interested in the case $t = min(i,j)$. This is because a $min(i,j)$-path has the important property that it is either a direct edge or is composed of a $min(i,k)$-path and a $min(k,j)$-path for some $k < min(i,j)$. Indeed, choose k to be the maximum node index along the $min(i,j)$-path and consider the sub-path from i to k. By choice, k is larger than any node index in this sub-path (excluding i) and consequently this is a $min(i,k)$-path from i to k. Likewise, the k to j path is also a $min(k,j)$-path, thereby verifying our statement. Clearly, the converse statement is also true. That is, if there is a $min(i,k)$-path from i to k and a $min(k,j)$-path from k to j for some $k < min(i,j)$, or if there is a directed arc from i to j, then there is a $min(i,j)$-path from i to j. Writing this up algebraically, we have

$$\mathbf{c}_{ij} = \mathbf{a}_{ij} \vee \left\{ \bigvee_{k=1}^{min(i,j)-1} \mathbf{c}_{ik} \wedge \mathbf{c}_{kj} \right\} \qquad (2)$$

which is merely a mathematical formulation of our earlier statement. The above equation clearly shows that the element c_{ij} of the matrix C is dependent only upon elements that are in the same column above it and in the same row to the left of it. (Note its similarity with the so-called 'quarter-plane' two-dimensional recursive filter [13-14].) Thus, the computation of these elements can be performed iteratively starting from the top left hand corner of the matrix C and proceeding downwards and outwards.

The existence and uniqueness of the solution to the set of equations in (2) can be easily established as shown in

[19]. The problem of determining the existence of a $min(i,j)$-path from node i to node j for every (i,j) is, therefore, quite easy to solve. Moreover, as shown below, this problem can be solved by means of a regular iterative algorithm; thus the techniques developed in [3,4] for analyzing these algorithms and for implementing them on mesh connected processor arrays can be applied. Later in this section, we shall show how to extend the given graph using additional N nodes (and some arcs) in such a manner that by using the information about the existence of $min(i,j)$-paths in this $2N$-node extended graph, one can infer the transitive closure of the original graph.

a. Determining the Min(i,j)-paths of a Directed Graph

The system of equations for determining the $min(i,j)$-paths of a directed graph, given in (2) above, resembles in a modified, simplified form, the set of relations that is usually associated with 'quarter-plane' recursive filtering in the signal processing literature [13-14]. Since this filtering problem can be solved by means of a regular iterative algorithm [4], it should be possible to determine the $min(i,j)$-paths of the graph using an algorithm of the same form. Such an algorithm is given below:

Algorithm 1: for determining the min(i,j)-paths of a directed graph: For $i,j,k = 1$ to N do:

$$a(i,j+1,k) = \begin{array}{l} a(i,j,k) \text{ if } j \neq k \\ c(i,j,k) \vee a(i,j,k) \cdot b(i,j,k) \text{ if } j = k \end{array}$$

$$b(i+1,j,k) = \begin{array}{l} b(i,j,k) \text{ if } i \neq k \\ c(i,j,k) \vee a(i,j,k) \cdot b(i,j,k) \text{ if } i = k \end{array}$$

$$c(i,j,k+1) = c(i,j,k) \vee [a(i,j,k) \cdot b(i,j,k)] \tag{3}$$

with the initialization

$$\begin{aligned} a(i,1,k) &= 0 \\ b(1,j,k) &= 0 \\ c(i,j,1) &= \mathbf{a}_{ij} \end{aligned} \tag{4}$$

for all $i,j,k = 1$ to N. Then $c(i,j,N+1) = 1$ if there is a $min(i,j)$-path from node i to node j and is equal to zero otherwise.

Some Variations: It is possible to relate the above algorithm to the 'first pass' of the systolic algorithm in [1]. However, in [1], the adjacency matrix of the graph is input twice to the array, once from the left end of the array and once from the top of the array. In the above algorithm, the adjacency matrix is input only once, i.e., by initializing $c(i,j,1)$. This would indicate that there may be other possible initializations for the algorithm that achieve the same objective. For example, the initialization of the systolic array in [1] corresponds to the following:

$$\begin{aligned} a(i,1,k) &= \mathbf{a}_{ik} \\ b(1,j,k) &= \mathbf{a}_{kj} \\ c(i,j,1) &= 0 \end{aligned} \tag{5}$$

where it is assumed that $\mathbf{a}_{ii} = 1$. We can show that for this initialization, or if in eqn.(5) $c(i,j,1)$ is also made equal to $\mathbf{a}_{ij}$, the algorithm can be used to obtain more information than just the $min(i,j)$-paths in the graph. In the following theorem, a weaker version of which appears in [2], the nature of the paths detected by the algorithm for this initialization is described.

Theorem 1: If Algorithm 1 is executed with the initialization of eqn. (5), then $c(i,j,N+1)$ will be 1 if either

i) there is a $min(i,j)$-path from node i to node j, or

ii) if there is a path of length 2 from node i to node j, or

iii) for $i > j$, if there is a decreasing index path from i to some k, where $i > k > j$, and a $min(k,j)$-path from k to j, or

iv) for $j > i$, if there is a $min(i,k)$-path from i to some k, where $j > k > i$, and an increasing index path from k to j.

If none of these paths exist from node i to node j, then $c(i,j,N+1)$ will be zero.

In contrast to the initialization used in Algorithm 1, the above initialization allows us to obtain more information than just the $min(i,j)$-paths in the graph. Nevertheless, this extra information is never used in the algorithms described in the rest of this section. Thus, if one is interested only in the $min(i,j)$-paths, in both Algorithm 1 and in the above variation, one could consider $c(i,j,min(i,j)+1)$ as the outputs of the algorithm. Furthermore, for determining $min(i,j)$-paths in the graph, one can devise a whole variety of possible initializations of Algorithm 1. To be specific,

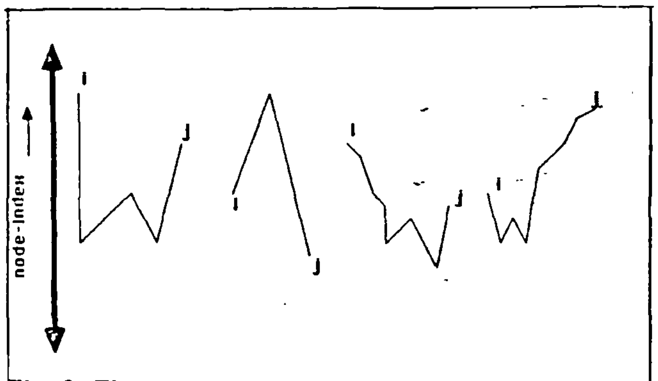


Fig. 2: The types of paths detected by Algorithm 1 with the initialization of eqn. (5): The nodes are assumed to be arranged in descending order from top to bottom (i.e. N appears at the top and 1 at the bottom). The paths shown are from left to right: *i)* $min(i,j)$-path, *ii)* path of length 2, *iii)* a path comprised of a decreasing index tail and a $min(i,j)$-path, and *iv)* a path comprised of a $min(i,j)$-path followed by an increasing index tail.

Theorem 2: If the inputs to Algorithm 1 are chosen such that

$$\begin{aligned} a(i,1,j) \cdot c(i,j,1) &= \mathbf{a}_{ij} \quad \text{for } i > j \\ b(1,j,i) \cdot c(i,j,1) &= \mathbf{a}_{ij} \quad \text{for } j > i \\ c(i,i,1) \cdot [a(i,1,i) \cdot b(1,i,i)] &= 1 \end{aligned} \tag{6}$$

then $c(i,j,min(i,j)+1)$ will be 1 if there is a $min(i,j)$-path from node i to node j and will be zero otherwise.

Theorem 2 opens up several interesting possibilities. Thus, for example, in the systolic array of Guibas, Kung and Thompson[1], one does not necessarily have to input the adjacency matrix twice, as described by the authors. Instead, again as an example, the lower triangular portion of the adjacency matrix can be input from the top of the array and the upper triangular portion from the left of the array. These and other variations are discussed in detail in Section IV of this paper.

b. The Extended Graph Approach to the Transitive Closure Problem

Knowing how to determine the $min(i,j)$-paths in a graph using a regular iterative algorithm, we can now solve the transitive closure problem itself. To do so, we shall attempt to extend the graph in such a manner that, solving the $min(i,j)$-path problem for this extended graph is equivalent to solving the transitive closure problem for the original graph. Even though this idea of extending the domain of the problem is borrowed from the signal processing literature, the basis for this approach can be independently derived using the following simple observation:

Given a directed graph with N nodes, suppose that we introduce additional N nodes numbered from $(N+1)$ through $2N$ such that node $(N+i)$ has one incoming arc from node i and one outgoing arc to node i and no other arcs incident upon it. For this modified graph, we claim that if there is a $min(N+i,N+j)$-path from node $(N+i)$ to node $(N+j)$, then and only then is there a path in the original graph from node i to node j. This statement is easily verified since all paths between these two nodes have to pass through node i and node j, and since every path in the original graph between node i and node j has to be part of a $min(N+i,N+j)$-path between node $(N+i)$ and node $(N+j)$.

The $2N$-node extended graph described above has an adjacency matrix $\hat{\mathbf{A}}$, given by

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & \mathbf{I} \\ \mathbf{I} & \mathbf{I} \end{bmatrix} \tag{7}$$

where $\mathbf{I}$ is the $(N \times N)$ identity matrix.

This is a rather simple extension of the original graph and was derived using heuristic arguments. Instead, one can pose the the extension problem formally as follows:

The Extension Problem: Given an N-node directed graph with adjacency matrix $\mathbf{A}$, determine an extension of this graph with adjacency matrix

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \tag{8}$$

with appropriate $(N \times N)$ matrices $\mathbf{A}_{12}$, $\mathbf{A}_{21}$ and $\mathbf{A}_{22}$ such that for $i,j \leq N$, there is a $min(N+i,N+j)$-path from node $(N+i)$ to node $(N+j)$ in the extended graph if and only if there is a directed path from node i to node j in the original graph.

Characterization of Useful Solutions to the Extension Problem:

An useful solution to the extension problem is one that can be 'easily' inferred from the original graph. Here, we shall assume that the extension can be easily inferred if it can be deduced from the adjacency matrix of the graph or from the $min(i,j)$-paths in the graph. Such a family of solutions to the extension problem can be characterized as follows:

Theorem 3: A graph with adjacency matrix given as in eqn. (17) is a solution to the extension problem if

$$\mathbf{A}_{12}\mathbf{I} = \mathbf{A}_{12}$$
$$\mathbf{A}_{21}\mathbf{I} = \mathbf{A}_{21}$$
$$\mathbf{A}_{22}\mathbf{I} = \mathbf{A}_{22} \tag{9}$$

and

$$\mathbf{A}_{12}\mathbf{T} = \mathbf{A}_{21}\mathbf{T} = \mathbf{A}_{22}\mathbf{T} = \mathbf{T} \tag{10}$$

where the OR of two matrices indicates the term by term logical OR of their entries. Further, in this extended graph, the existence of a path from node $(N+i)$ to node $(N+j)$ implies the existence of a $min(N+i,N+j)$-path from node $(N+i)$ to node $(N+j)$.

The extended graph can be chosen in several ways. For instance, the choice $\mathbf{A}_{12} = \mathbf{A}_{21} = \mathbf{A}_{22} = \mathbf{I}$ that we discussed earlier, satisfies the conditions in Theorem 3. Likewise, the choice $\mathbf{A}_{12} = \mathbf{A}_{21} = \mathbf{A}_{22} = \mathbf{A}$ is also valid. Alternately, we could first determine the $min(i,j)$-paths in the original graph and use this information for initializing the extended graph. Note that once the extended graph is chosen, Algorithm 1 or any of its variations can be applied on the extended graph so as to obtain the transitive closure of the original graph, even though some of the variations described earlier determine more than just the $min(i,j)$-paths in the graph. This is because of the fact (stated in Theorem 3) that for every path in the extended graph from node $(N+i)$ to node $(N+j)$, there exists a corresponding min-path. Furthermore, since we are only interested in the $min(N+i,N+j)$-paths, the transitive closure information is also obtained as $c(N+i,N+j,N+min(i,j)+1) = \mathbf{t}_{ij}$.

The above statements were made without taking into consideration the structure of the extended graph. Using the structure in the graph, one would hope that determining the existence of $min(N+i,N+j)$-paths from node $(N+i)$ to node $(N+j)$ is 'easier' in some sense. Indeed this is true due to the following result:

Theorem 4: If Algorithm 1 is executed on any member of the family of extended graphs characterized in Theorem 3, then

a) For $i,j \leq N$, if $i > j$, and if there exists a $max(i,j)$-path from node i to node j in the original graph, then $c(i,N+j,i+1) = 1$.

b) For $i,j \leq N$, if $j > i$, and if there exists a $max(i,j)$-path from node i to node j in the original graph, then $c(N+i,j,j+1) = 1$.

c) Finally, $c(N+i,N+j,N+1) = \mathbf{t}_{ij}$, for all $i, j \leq N$.

With this theoretical background established, we can state the basic algorithm for determining the transitive closure of the given graph using the extended graph approach. Later, we shall particularize the algorithm to certain special cases (and deduce some variations) so as to obtain, e.g., the algorithms reported in [1,2].

Algorithm 2:(for determining the transitive closure of a directed graph):

i. Choose an extension of the graph using Theorems 3,4.
ii. Execute Algorithm 1 for the extended graph with adjacency matrix $\hat{\mathbf{A}}$ and over the index space $i,j = 1$ to $2N$ and $k = 1$ to $min(i,j,N)$.
iii. Determine the transitive closure, $\{\mathbf{t}_{ij}\}$, of the original graph as $\{\, c(N+i,N+j,N+1)\}$ for all $i,j \leq N$.

c. Some Variations of Algorithm 2 and Some Special Graph Extensions

Given the basic algorithm for solving the transitive closure problem, *viz.* Algorithm 2, one can easily come up with variations based on some simple observations. In this manner, we shall presently show how to deduce the algorithms given in [1] and [2] as special cases of the extended graph approach.

Suppose that Algorithm 2 is executed and the values of $c(i,j,min(i,j,N)+1)$ are collected in matrix form in the obvious order, for $i,j = 1$ to $2N$. Call the resulting $(2N \times 2N)$ matrix $\hat{\mathbf{C}}$, and partition it into $(N \times N)$ blocks as

$$\hat{\mathbf{C}} = \begin{bmatrix} \mathbf{C} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix} \tag{11}$$

Then, from Theorem 4, it is known that

$$\mathbf{C}_{22} = \mathbf{T} \tag{12}$$

and

$$\mathbf{C}_{21} \times \mathbf{C}_{12} = \mathbf{T} \tag{13}$$

where the symbol '×' signifies the logical product of the two matrices. (The logical product is the same as the usual matrix product except that scalar addition is replaced by the logical *OR* operation and scalar multiplication by the logical *AND* operation.) Furthermore, we know that

$$\mathbf{C}_{-:}\cdot\mathbf{T} = \mathbf{C}_{::}\cdot\mathbf{T} = \mathbf{T} \tag{14}$$

and from the definition of the transitive closure matrix,

$$\mathbf{T}\times\mathbf{T} = \mathbf{T} \tag{15}$$

Therefore, one can show that

$$\left\{\mathbf{C}_{-:}\cdot\mathbf{C}_{::}\right\}\times\left\{\mathbf{C}_{-:}\cdot\mathbf{C}_{::}\right\} = \mathbf{T} \tag{16}$$

Now, in Algorithm 2, if it is applied to the extended graph without modification, one evaluates $\mathbf{C}_{-:}$ and $\mathbf{C}_{::}$ separately and then finds the logical product of the two to determine the transitive closure of the original graph. However, eqn.(16) implies that we do not need to determine these matrices separately. The logical *OR* of these matrices is sufficient for our purpose. This observation is, in spirit, the basis of the systolic transitive closure algorithm of Guibas, Kung and Thompson[1], though these authors do not derive their algorithm using such considerations.

Certain choices of the extended graph are 'natural' and need to be examined in some detail. Thus, the choice $\mathbf{A}_{-:} = \mathbf{A}_{:-} = \mathbf{A}_{::} = \mathbf{I}$, discussed previously, is a simple and straight-forward solution to the extension problem. For such a choice, the matrix $\hat{\mathbf{C}}$, defined above, is such that $\mathbf{C}_{-:}$ is lower triangular with ones on the diagonal, and $\mathbf{C}_{::}$ is upper triangular with ones on the diagonal. This fact can be easily verified using the observation that $c(N+i,j,j+1)$ is 1 if there is a $min(N+i,j)$-path from node $(N+i)$ to node j in the extended graph and is 0 otherwise. Now, for the above choice of the extended graph, any path from node $(N+i)$ to node j has to pass through node i and therefore, if $i > j$, all these paths are not $min(N+i,j)$-paths. Hence $c(N+i,j,j+1) = 0$ for $i > j$, which substantiates our statement that $\mathbf{C}_{::}$ is upper triangular. Similarly, one can conclude that $\mathbf{C}_{-:}$ is lower triangular.

For this particular choice of the extended graph, the matrix $\mathbf{C}_{-:}$ is lower triangular while the matrix $\mathbf{C}_{::}$ is upper triangular. However, for any arbitrary choice of the extension, these matrices need not be triangular. Nevertheless, we can show that the relevant triangular portions of the resulting matrices are of interest because of the following theorem.

Theorem 5: For any choice of the extended graph satisfying Theorem 3, let

$$\mathbf{C}_{-:} = \mathbf{L}_{-:}\cdot\mathbf{I}\cdot\mathbf{U}_{-:}$$

$$\mathbf{C}_{::} = \mathbf{L}_{::}\cdot\mathbf{I}\cdot\mathbf{U}_{::} \tag{17}$$

where the L's are lower triangular and U's are upper triangular matrices. Define

$$\mathbf{L} = \mathbf{L}_{-:}\cdot\mathbf{I}$$

$$\mathbf{U} - \mathbf{U}_{::}\cdot\mathbf{I} \tag{18}$$

Then, the transitive closure of the original graph, $\mathbf{T}$ is given by

$$\mathbf{T} = \mathbf{U}\times\mathbf{L} \tag{19}$$

Notice how the structure of the extended graph has been used to reduce the computational effort involved in Algorithm 2. Thus, in Theorem 4, we showed that instead of considering $c(i,j,min(i,j)+1)$ for the transitive closure of the graph, one can instead use $c(i,j,min(i,j,N)+1)$. This reduces the total amount of computation by approximately $N^3/3$ logical *AND* and logical *OR* operations. Next, in Theorem 5, we showed that we do not need to compute the matrices $\mathbf{C}_{-:}$ and $\mathbf{C}_{::}$ entirely. Only the lower triangular portion of $\mathbf{C}_{-:}$ and the upper triangular portion of $\mathbf{C}_{::}$ are of relevance. This further reduces the total computational effort by about $N^3/3$ logical operations. Thus, using these theorems, we have reduced the computational effort from $8N^3/3$ operations to $2N^3$ operations. Nevertheless, for a sequential implementation, this is still about a factor of 2 greater than the computational effort required for Warshall's algorithm[10]. However, Warshall's algorithm can be derived using the extended graph approach by considering the graph extension for which $\mathbf{A}_{-:} = \mathbf{A}_{:-} = \mathbf{A}_{::} = \mathbf{A}$. An examination of the operation of Algorithm 2 for this extension of the graph reveals that some of the intermediate variables computed by the algorithm are identical and therefore the computation of these variables need not be repeated. In the algorithm, $c(N+i,N+j,N+1)$ is obtained by computing iteratively for $k = 1$ to N,

$$\begin{aligned} c(N+i,N+j,k+1) = {} & c(N+i,N+j,k)\vee \\ & [a(N+i,N+j,k)\wedge b(N+i,N+j,k)] \end{aligned} \tag{20}$$

Now, the variables $a(N+i,N+j,k)$ and $b(N+i,N+j,k)$ in eqn. (20) are really dummy variables that represent $c(N+i,k,k+1)$ and $c(k,N+j,k+1)$ respectively. However, for the special choice of the extended graph under consideration, we can show after some algebraic manipulations, that $c(N+i,k,k+1) = c(N+i,N+k,k)$ and $c(k,N+j,k+1) = c(N+k,N+j,k)$ and therefore,

$$\begin{aligned} c(N+i,N+j,k+1) = {} & c(N+i,N+j,k)\vee \\ & [c(N+i,N+k,k)\wedge c(N+k,N+j,k)] \end{aligned} \tag{21}$$

This is precisely the transitive closure algorithm due to Warshall[10] (a shortest-path version of the algorithm is due to Floyd[11] and a minor modification of the algorithm, in its shortest path version, is attributed to Dantzig[15]), which was recently implemented in a systolic array by Kung and Lo[2].

Clearly, for a sequential implementation, the above algorithm requires N^3 logical *OR* and logical *AND* operations which implies a factor of two reduction in computational effort. Nevertheless, the amount of parallel time (from first input to last output) required for the $(N\times N)$ systolic array in [1] and the $(N\times N)$ systolic array in [2] is the same and equal to $5N$. This is because Warshall's algorithm requires forward and backward processing of the data (both increasing and decreasing i,j indices) whereas in the general case, in Algorithm 2, the data is always propagated in the forward (increasing i,j indices) direction. Thus, in order to implement Warshall's algorithm on a systolic array, the authors in [2] had to introduce a feed-back loop which renders the *iteration interval*, i.e., the time between successive data samples, to be 3, whereas the iteration interval for the array in [1] is 1. Thus, the total time taken is the same in both cases, even though the systolic array in [1] requires three passes for computing the transitive closure, whereas the array in [2] requires only one pass.

Implementation of the Regular Iterative Algorithm

Following the procedure given in [3,17], let us define a number of 'parallel iteration steps' for the algorithm in the following fashion: at iteration step t, variables $a(i,j,k)$, $b(i,j,k)$ and $c(i,j,k)$ will be computed if $i+j+k = t$. The expression $(i+j+k)$ is the so-called *linear scheduling function* for these variables [3], since it maps each variable at any index point into a unique iteration step. Furthermore, the computation of the variables at iteration step t requires, as input, the values of variables that are computed at iteration step $t-1$, since *e.g.*, $a(i,j,k)$ is dependent upon $a(i,j-1,k)$ and so on. Therefore, if we can schedule the computation of the algorithm such that all variables at step $t-1$ are computed before we proceed to step t, then all precedence constraints will be satisfied.

Having obtained the linear scheduling functions for the algorithm, we can now proceed to systematically implement Algorithm 2 on a mesh-connected processor array using the techniques described in [4,17]. To do so, note that the index-space for Algorithm 2, i.e. the span of the indices $\{i,j,k\}$ for the variables in the algorithm, can be described by the set of constraints

$$1 \le i \le N$$
$$[\, max(i,N)-N+1 \,] \le j \le [\, min(N,i)+N \,]$$
$$1 \le k \le [\, min(i,j,N) \,] \qquad (22)$$

where the variation of Algorithm 2, suggested by Theorem 5 is assumed. Thus, a streamlined version of Algorithm 2 has the index space shown in Fig.3. Any point with integer coordinates (i,j,k), that lies within this space, is referred to as an index point. The index point (i,j,k) conceptually represents the computation described by the iteration unit of the algorithm. Thus, at this index point, the variables $a(i,j,k)$, $b(i,j,k)$ and $c(i,j,k)$ are made available to the algorithm and after some time, the variables $a(i,j+1,k)$, $b(i+1,j,k)$ and $c(i,j,k+1)$ are computed and sent to the appropriate index points.

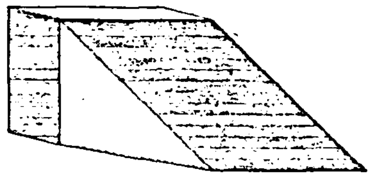


Fig.3: The index space for a variation of Algorithm 2: The three dimensional solid shown above is specified by the set of constraints in eqn.(22).

Just as an example, let us now assume that all variables with the same k-index are executed by the same processor. In terms used in [3,4], the *iteration-space* is [0 0 1]. This defines a two-dimensional processor array (shown in Fig.4) with approximately $3N^2$ processors. In this array, the a and b values are propagated to the neighbouring processors while $c(i,j,k)$ represents a register value that is updated within processor at location (i,j).

In the array shown in Fig.4, the processor at location (i,j) computes all variables with index (i,j,k) at 'time' $(i+j+k)$. Using this fact and by examining the set of constraints that determine the index-space, one can argue that the processors at locations (i,j), $(i+N,j)$ for $i \le j \le N$, $(i,j+N)$ for $j \le i \le N$, and at location $(i+N,j+N)$ are active during disjoint intervals of time. The operations of all these processors can therefore be simulated on a single one; the resulting array, shown in Fig.5, is precisely the array derived in [1].

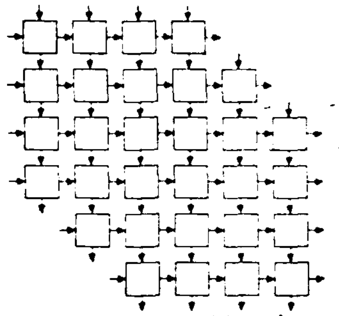


Fig.4: A processor array with $3N^2$ processing elements: This array is obtained for the iteration space [0 0 1].

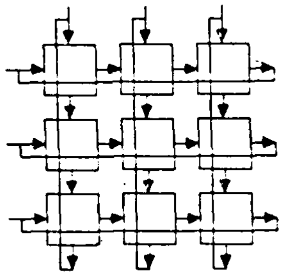


Fig. 5: The processor array obtained by multiplexing disjointly active processors in the array of Fig.4: For a particular choice of the extended graph, this array reduces to that described in reference [1].

References

[1] L.J. Guibas, H.T. Kung and C.D. Thompson, 'Direct VLSI Implementation of Combibnatorial Algorithms,' *Proc. Cal-Tech Conference on VLSI*, 1979.

[2] S.Y. Kung and S.C. Lo, 'A Fast Systolic Algorithm for Transitive Closure Problem,' *Proc. of the ICASSP*, 1985.

[3] S.K. Rao, H. V. Jagadish and T. Kailath, 'Analysis of Iterative Algorithms for Multiprocessor Implementations, *submitted to the IEEE Trans. on Circuits and Systems*, 1985.

[4] H.V. Jagadish, S.K. Rao and T. Kailath, 'Multi-Processor Architectures for Iterative Algorithms,' *submitted to Proceedings of the IEEE*, 1985.

[5] J.D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Rockville, MD, 1984.

[6] R.C. Backhouse and B.A. Carre, 'Regular Algebra Applied to Path-Finding Problems,' *J. Inst. of Math. Appl.*, 1975.

[7] B.A. Carre, *Graphs and Networks*, Clarendon Press, Oxford, 1979.

[8] S.K. Abdali and B.D. Saunders, 'Transitive Closure and Related Semiring Properties via Eliminants, *to appear in Theoretical Computer Science*, 1985.

[9] L.R. Ford and D.R. Fulkerson, *Flows in Networks*, Princeton University Press, 1962.

[10] S. Warshall, 'A Theorem on Boolean Matrices,' *J. ACM*, 1962.

[11] R.W. Floyd, 'Algorithm 97: Shortest Path,' *C. ACM*, 1962.

[12] L. Ljung and T. Kailath, 'A Unified Approach to Smoothing Formulas,' *Automatica*, 1976.

[13] D.E. Dudgeon and R.M. Mersereau, *Multidimensional Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1984.

[14] W.K. Pratt, *Digital Image Processing*, John Wiley and Sons, New York, 1978.

[15] G.B. Dantzig, 'All Shortest Routes in a Graph', *Operations Research House*, Stanford University Tech. Rep. No. 66-3, Nov. 1966.

[16] J.J. Baker, 'A Note on Multiplying Boolean Matrices,' *C. ACM*, 1962.

[17] S.K. Rao, 'Regular Iterative Algorithms and their Implementations on Processor Arrays,' *Ph. D. Dissertation, ISL, Stanford University, Stanford, CA94305, Sept. 1985.*

[18] R.M. Karp, R.E. Miller and S. Winograd, 'The Organization of Computations for Uniform Recurrence Equations,' *J.ACM*, Jul.1967.

[19] S.K. Rao, T.K. Citron and T. Kailath, 'Mesh-Connected Processor Arrays for the Transitive Closure Problem,' *Submitted to J. ACM*, 1985.

END

DTIC

7-86